# NAG Library Routine Document

# M01DFF

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

## 1 Purpose

M01DFF ranks the rows of a matrix of integer numbers in ascending or descending order.

## 2 Specification

```
SUBROUTINE M01DFF (IM, LDM, M1, M2, N1, N2, ORDER, IRANK, IFAIL)
INTEGER      IM(LDM,N2), LDM, M1, M2, N1, N2, IRANK(M2), IFAIL
CHARACTER(1) ORDER
```

## 3 Description

M01DFF ranks rows M1 to M2 of a matrix, using the data in columns N1 to N2 of those rows. The ordering is determined by first ranking the data in column N1, then ranking any tied rows according to the data in column N1 + 1, and so on up to column N2.

M01DFF uses a variant of list-merging, as described on pages 165–166 in Knuth (1973). The routine takes advantage of natural ordering in the data, and uses a simple list insertion in a preparatory pass to generate ordered lists of length at least 10. The ranking is stable: equal rows preserve their ordering in the input data.

## 4 References

Knuth D E (1973) *The Art of Computer Programming (Volume 3)* (2nd Edition) Addison–Wesley

## 5 Arguments

1: IM(LDM, N2) – INTEGER array *Input*

*On entry*: columns N1 to N2 of rows M1 to M2 of IM must contain integer data to be ranked.

2: LDM – INTEGER *Input*

*On entry*: the first dimension of the array IM as declared in the (sub)program from which M01DFF is called.

*Constraint*: LDM $\geq$ M2.

3: M1 – INTEGER *Input*

*On entry*: the index of the first row of IM to be ranked.

*Constraint*: M1 $>$ 0.

4: M2 – INTEGER *Input*

*On entry*: the index of the last row of IM to be ranked.

*Constraint*: M2 $\geq$ M1.

5: N1 – INTEGER *Input*

*On entry*: the index of the first column of IM to be used.

*Constraint*: N1 $> 0$.

6: N2 – INTEGER *Input*

*On entry*: the index of the last column of IM to be used.

*Constraint*: N2 $\geq$ N1.

7: ORDER – CHARACTER(1) *Input*

*On entry*: if ORDER = 'A', the rows will be ranked in ascending (i.e., nondecreasing) order.

If ORDER = 'D', into descending order.

*Constraint*: ORDER = 'A' or 'D'.

8: IRANK(M2) – INTEGER array *Output*

*On exit*: elements M1 to M2 of IRANK contain the ranks of the corresponding rows of IM. Note that the ranks are in the range M1 to M2: thus, if the $i$th row of IM is the first in the rank order, IRANK($i$) is set to M1.

9: IFAIL – INTEGER *Input/Output*

*On entry*: IFAIL must be set to 0, $-1$ or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value $-1$ or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this argument, the recommended value is 0. **When the value $-1$ or 1 is used it is essential to test the value of IFAIL on exit.**

*On exit*: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

## 6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or $-1$, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, M2 $< 1$,
or N2 $< 1$,
or M1 $< 1$,
or M1 $>$ M2,
or N1 $< 1$,
or N1 $>$ N2,
or LDM $<$ M2.

IFAIL $= 2$

On entry, ORDER is not `A' or `D'.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

## 7 Accuracy

Not applicable.

## 8 Parallelism and Performance

M01DFF is not threaded in any implementation.

## 9 Further Comments

The average time taken by the routine is approximately proportional to $n \times \log(n)$, where $n = \mathrm{M2} - \mathrm{M1} + 1$.

## 10 Example

This example reads a matrix of integers and ranks the rows in descending order.

### 10.1 Program Text

```
    Program m01dffe

!     M01DFF Example Program Text

!     Mark 26 Release. NAG Copyright 2016.

!     .. Use Statements ..
      Use nag_library, Only: m01dff
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter               :: nin = 5, nout = 6
!     .. Local Scalars ..
      Integer                          :: i, ifail, j, ldm, m1, m2, n1, n2
!     .. Local Arrays ..
      Integer, Allocatable             :: im(:,:), irank(:)
!     .. Executable Statements ..
      Write (nout,*) 'M01DFF Example Program Results'

!     Skip heading in data file
      Read (nin,*)

      Read (nin,*) m2, n2
      ldm = m2
      Allocate (im(ldm,n2),irank(m2))

      m1 = 1
      n1 = 1

      Do i = m1, m2
        Read (nin,*)(im(i,j),j=n1,n2)
      End Do

      ifail = 0
      Call m01dff(im,ldm,m1,m2,n1,n2,'Descending',irank,ifail)
```

```
      Write (nout,*)
      Write (nout,*) 'Data                          Ranks'
      Write (nout,*)

      Do i = m1, m2
        Write (nout,99999)(im(i,j),j=n1,n2), irank(i)
      End Do

99999 Format (1X,3I7,I11)
    End Program m01dffe
```

## 10.2 Program Data

```
M01DFF Example Program Data
12 3
6 5 4
5 2 1
2 4 9
4 9 6
4 9 5
4 1 2
3 4 1
2 4 6
1 6 4
9 3 2
6 2 5
4 9 6
```

## 10.3 Program Results

```
 M01DFF Example Program Results

 Data                          Ranks

       6      5      4          2
       5      2      1          4
       2      4      9         10
       4      9      6          5
       4      9      5          7
       4      1      2          8
       3      4      1          9
       2      4      6         11
       1      6      4         12
       9      3      2          1
       6      2      5          3
       4      9      6          6
```